VMM Verification Methodology Manual 活用テクニック

赤星博輝

第5回（最終回） 大規模回路のための検証環境を作成する

今回は，VMM（Verification Methodology Manual）に関する総仕上げを行います．これまでの検証環境では，入力ベクタを作成する部分と，ライブラリの使用法について説明してきましたが，チェックに関してはほとんど触れてきませんでした．そこで，これまでに紹介した機能を使って，連載第2回（本誌2006年10月号，pp.139-147）で示した領域判定回路をチェックする検証環境を作成します．（筆者）

大規模・複雑化が進む最近の設計では，検証が大変になっています．このような状況について，ほかの分野でどうなっているかを考えてみるのも時には重要だと思います．

例えば，穴を掘るという作業を考えてみましょう（**図1**）．人はシャベルを持って穴を掘ることができます．しかし，これは掘れる穴の大きさに実質的な制限があります．大きな穴をシャベルだけで掘ることはできるのですが，効率の悪い作業になります．大きな穴を掘る場合に効率などを考えると，ショベルカーなどを使うことになります．このときに考えないといけないのは，ショベルカーを使うには免許が必要だし，ショベルカーを所持またはレンタルするコストもかかることです．

これを，検証に当てはめてみます．検証対象が小規模で，個人でできるうちはコストは大してかかりません．その代わり，できる作業が限られます．大規模な開発に対応するためには，ツールの導入が必要であり，そのために教育を行い，ツールや人員のレベル維持に費用が発生するということになります．

ツールなどは必要となった時に購入すればよいのですが，教育に関しては地道な取り組みをしておかないと，そう簡単にレベルアップできないところに注意しておく必要があります．

1．チェックの自動化を検討する

連載第2回（本誌2006年10月号，pp.139-147）の領域判定回路は，**図2**のように2次元（256×256）の領域に対して，X軸が50～100，Y軸が70～120の間に入っていれば‘1’と判定し，それ以外は‘0’と判定する回路でした．

これまでは，ランダム生成を用いて入力パターンを発生させて，波形ビューワで確認するところで終わっていました．しかし，現実にランダム生成で大量のパターンを自動

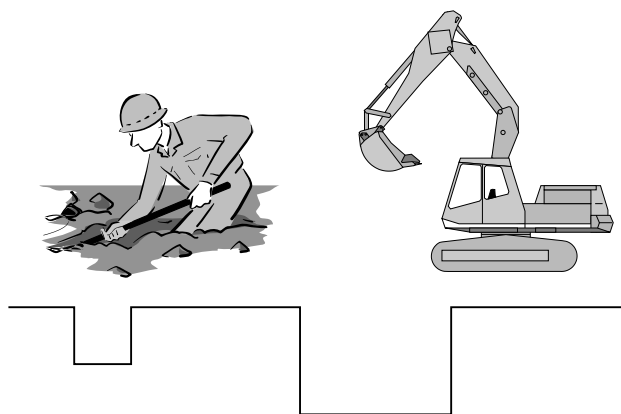



図1 世の中の動きと検証を対比して考える
やることが大きくなってくると，設備の導入とそれに伴う教育が重要になってくる．

KeyWord VMM，検証，スコアボード，リファレンス・モデル，ランダム生成，テストベンチ，キュー，コールバック，レポート

発生させると，設計者が波形を見て動作確認を行うのは実質的に不可能です．

それでは，ランダム生成で作成した入力パターンをどのようにチェックすればよいでしょうか．ランダム生成を使うことが前提のVMMではスコアボードというチェックの自動化が必須になってきます．

● リファレンス・モデルを利用する

チェックの自動化は，どのように実現すればよいでしょうか．

これは，状況によって変わってきます．例えば，C/C++言語で詳細にアルゴリズムなどの検討を行っている場合には，**図3**のように，C/C++言語のモデルをリファレンスとして使用することができます．このように，あらかじめ検証済みのリファレンス・モデルがあればそれを利用できます．

リファレンスになるものがない場合には，どうするか考える必要があります．

ひとつのやり方として，設計とは別にリファレンス・モデルを開発する方法があります．このリファレンス・モデルの作成は，C言語でもHDLでも開発可能です．さらに合成を意識する必要がないので，合成を行うHDL記述よりも短い時間で作成できます．

しかし，いくつか問題点があります．

まず，同時に二つのモデル（設計とリファレンス）を開発するため，多くの開発リソースを必要とするという問題があります．最近の開発では，特に人的リソースが不足気味なので，なかなかこの手法が取り難い面があります（開発プランをうまく立て，うまくスケジューリングするなどの必要がある）．

また，リファレンス・モデルの作り方が問題になることがあります．例えば，検証用のリファレンス・モデル作成時に，設計側のコードと共用にすることがあります． 領域判定回路の検証で考えてみると，**図3**で判定する関数を共用しているような場合です．これもリファレンス・モデルだけで検証が済んでいればよいのですが，そうでないと問題が残っている場合があります．共用した場所にバグがある場合には，リファレンス・モデルとDUT（device under test）とを比較していても，バグを検出できません．

● ランダム生成を活用する

こういった時にはアイデアが必要です．実はランダム生成を工夫すると検証がやりやすくなる場合もあります．これまで領域判定の検証では，2次元の座標をランダム生成し，その座標値をドライバがDUTに投入していました．これを**図4**のように，領域内（HIT）と領域外（MISS）という二つの状態についてランダム生成を行い，次に，そのHITなら領域内の値でランダム生成し，MISSなら領域外の値でランダム生成します．このHITかMISSという情報をスコアボードに期待値として渡すことで，スコアボードには

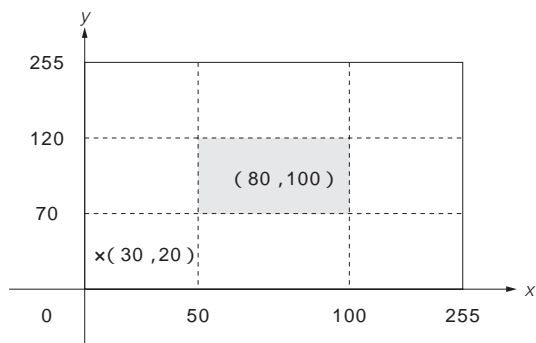



図2 領域判定問題
256×256の領域で，50≦x≦100，70≦y≦120に入っているか，入っていないかを判定する．

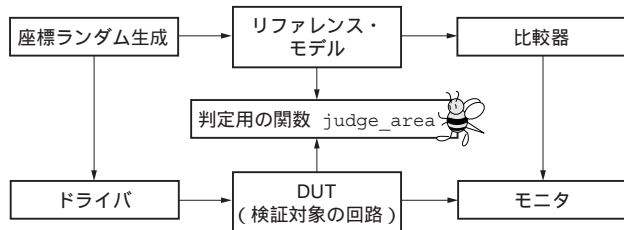



図3 リファレンス・モデルが使用できる場合
ランダム生成した値をリファレンス・モデルを使い期待値を作成し，その結果を比較することが可能になる．

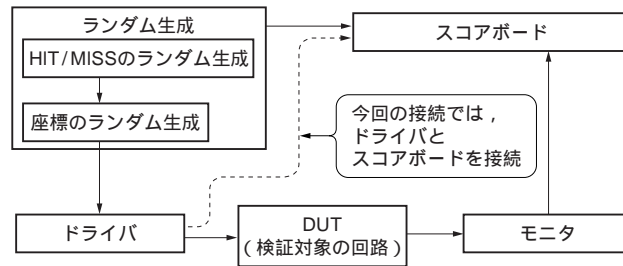



図4 ランダム生成を活用し期待値比較を容易に
ランダム生成も発想を変えると，スコアボードで簡単に出力値のチェックが可能となる．

期待値としてHITまたはMISSという情報が与えられます．また，モニタからは出力値として範囲に入っている場合は‘1’，そうでない場合は‘0’に関する情報が送られてきます．期待値がHITの出力値が‘1’，期待値がMISSの場合には出力が‘0’であればPASSとなるので，スコアボードでは簡単にチェックできることが分かります．

このように，リファレンス・モデルの代わりにランダム生成を使うことで，検証側と設計側で同じコードを使用することなく，しかも，短い記述で検証を行うことが可能な場合もあります．

VMMのランダム生成の特徴としては，短いテストベンチ記述から，大量のテスト・パターンを生成できることがあります．また，ランダム生成で検証したいパターンが網羅できない場合でも，VMMのランダム生成vmm_atomic_genを使っておけば，injectを使うことでテストベンチの構造をほとんど再利用してダイレクト・テストベンチを作成できることがあります(本誌2007年1月号，pp.125-132の連載第3回を参照)．

このため，ランダム生成とダイレクトなテストベンチの両方をうまく使い，検証の効率化を図ることができます．

ランダム生成を使うと，テストベンチの作り方に関するバリエーションが増えるので，いろいろ使い方を考えてみるとおもしろいと思います．

2．領域判定回路をチェックする検証環境の作成

図4の環境を実現するために，データを扱うクラスのxy_datを**リスト1**のように変更しました(本来は，継承を使うことで差分だけ実装ができるが，今回は説明のために元のクラスを変更した)．HITとMISSのためにデータ型xy_kindをenumを用いて定義し，そのxy_kindの変数kindをランダム変数としての宣言を追加しました．スコアボードでは，このHITとMISSを使ってチェックを行います．そして制約として，変数kindがHITの場合は領域内の値，変数kindがMISSの場合には領域外の値とします．また，ランダム生成の順番の定義として，kindを最初に行い，その後にmX，mYに対して行うためのsolveという記述を加えました．このsolveはSystem Verilogで加えられた構文で，ランダム生成を効率的に行うための構文になります．

● 出力値を取り出すモニタの作成

スコアボードを作成する前に，モニタを作成します．ここでは，DUTの出力を監視し，出力があればその出力データを画面に出力するモニタを作成します．

このモニタは処理を行うので，VMMではvmm_xactorをベースに作成することになります(**リスト2**)．vmm_xactorではタスクmainで処理を記述しますが，その処理の最初にsuper.main()を呼び，その後にこのモニタで行う処理を記述します．

このxy_monitorでは，20回のデータを受信するため，タスクmainの中でfor文を使って20回receive_datを呼び出します．receive_datでは，DUTのoenが‘1’になったらojudgeの値を読み込み，その値が‘1’ならHIT，その値が‘0’ならMISSとし，その結果をxy_datに入れて返します(ここでは，mX，mYに255を入れた)．そのreceive_datから帰ってきた値を，mainのfor文中で画面にその結果を出力します．

リスト1
期待値チェックを意識したランダム生成
判定用の変数kindを追加し，そのkindに従って制約を与えてランダム生成を行う．

```
class xy_dat extends vmm_data;
 typedef enum {HIT,MISS} xy_kind;
 rand xy_kind    kind;

 rand logic[7:0] mX,mY;

constraint hit {
  if (kind == HIT)
      ( ( (50<=mX) && (mX<=100) )&& ( (70<=mY)&&(mY<=120) ) );

  if (kind == MISS)
     (! (( (50<=mX) && (mX<=100) )&& ( (70<=mY)&&(mY<=120) )));

  solve  kind before mX, mY;
 }
endclass
```

領域内(HIT)と領域外(MISS)のデータ型xy_kindを定義し，そのランダム変数kindを定義した

HITの場合には領域内でランダム生成，MISSの場合には領域外でランダム生成を行う

mX，mYのランダム生成の前に，kindのランダム生成を行う

リスト2
DUTのモニタを作成
タスクreceive_datでデータを受信し，受信したデータを出力する．これは，波形データをテキスト・データに変換する作業と考えるべき．

```
class xy_monitor extends vmm_xactor;
  function new(string inst,
                      int s_id=-1);
    super.new("monitor", inst, s_id);
  endfunction

  virtual task main();
      fork
      super.main();
    join_none
    for (int i=0; i<20; i++) begin
      xy_dat tmp=new;
      receive_dat(tmp);
      tmp.display("RECEIVE:");
    end
  endtask
```

(**a**) メイン・ルーチン

```
virtual task receive_dat(xy_dat tmp);
    @(posedge clk)
        #10
      while ( !uif.oen ) begin
          @(posedge clk);
      end
      #10
          if ( uif.ojudge == 1 )
            tmp.kind = xy_dat::HIT;
          else
            tmp.kind = xy_dat::MISS;
          tmp.mX = 255;
          tmp.mY = 255;
  endtask

endclass
```

(**b**) タスクreceive-dat

これは，ただ画面に結果を出して遊んでいるように見えますが，この作業はDUTの結果を波形からテキスト（文字列）に変換しています．波形のままでは自動化が困難ですが，テキストに変えてしまえば検証や解析などを自動化することが容易になります．

● スコアボードの作成手順

次に，出力値を自動でチェックするスコアボードを作成します．しかし，このスコアボードは，コールバックを使用したり，トランザクタ（vmm_xactor）やシミュレーション環境（vmm_env）などを使って行うことがあり，VMMで非常に複雑な部品の一つと言えます．

今回作成するスコアボードは，次のような機能を持ちます．

1）ドライバから期待値データを受け取る．
2）モニタからDUTの出力データを受け取る．
3）期待値と受信データを比較し判定する．
4）チェック結果に関するレポートを出力する．

このとき，ドライバやモニタとスコアボードをどう接続するのがよいでしょうか．これまでは，vmm_notifyやvmm_channelを紹介しましたが，これらは，スコアボードの接続に使用するのでしょうか．vmm_notifyはイベントを伝えるもので，データを伝えることはできません．それでは，スコアボードとドライバやモニタをvmm_channelを使って接続するかというと，それはあまり良くない選択になります．

検証したい項目や設計フェーズの進み方により，スコアボードの接続する場所が異なることを示したのが**図5**です．スコアボードを接続する場所が一つではないことが分かります．もし，vmm_channelでスコアボードで接続する部品を作成した場合には，スコアボードを使用しない場合にトラブルが発生します．vmm_channelはデフォルト・バッファのサイズが1になるので，スコアボードが接続されていない場合にはデータを1個投げ入れた段階でブロッキングされ動作が止まってしまいます．VMMではこうした場合には，コールバックを使うことになります．

VMMのvmm_xactorでは，任意の数のコールバックを追加できるようになっており，vmm_channelと違って必

(**a**) ランダム生成とモニタをスコアボードに接続した例

(**b**) ドライバとDUTをスコアボードに接続した例

(**c**) 処理Aと処理Xをスコアボードに接続した例

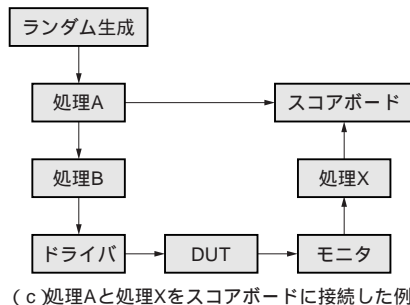
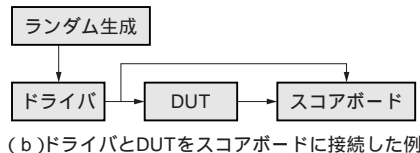
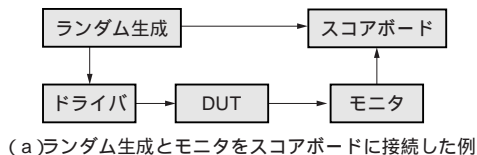



図5
スコアボードの接続場所
設計や検証が進むと，スコアボードを接続する場所が変わってくるので，vmm_channelでは対応が難しい．

要に応じて接続することができます．

基本的な考え方としては，常に接続する必要がある本質的なデータや制御の流れにのみvmm_channelを使用し，接続しない場合があるものはコールバックで対応できないか検討してみましょう．

今回は以下の手順でスコアボードを作成してみます．

1）スコアボード・クラスを作成する．
2）ドライバとモニタ用のコールバック用の仮想クラスをそれぞれ作成する．
3）ドライバとモニタにコールバック・ポイントを作成する．
4）スコアボード用のコールバックのクラスを作成する．
5）vmm_envで，スコアボード，ドライバ・コールバック，モニタ・コールバックを定義する．
6）vmm_envで，インスタンスを作成し，コールバックを登録する．
7）vmm_envで，スコアボード・クラスの起動を行う．
8）スコアボードのreportを呼び出す．

● スコアボード・クラスの作成

スコアボードは**図6**のような構成になります．基本的には，期待値を記憶するキュー（queue）と期待値を登録する関数，出力値をチェックする関数が必要です．今回は，内部チャネルvmm_xactorを使用するためのmainタスクと，チェック結果をレポートする関数を作成しました．

関数とタスクの使い分けは簡単です．時間を経過させない（ゼロ遅延）処理は関数で記述し，そうでない処理をタスクで記述します．

Verilog HDLなどでは，期待値を保存するためにはFIFOを使うことが多かったと思いますが，SystemVerilogには言語的にキューが使えます．そこで，**リスト3**のようにキューを定義し，期待値を登録する関数send_from_genが呼ばれると，push_backを使って期待値をキューに登録するようにしました．

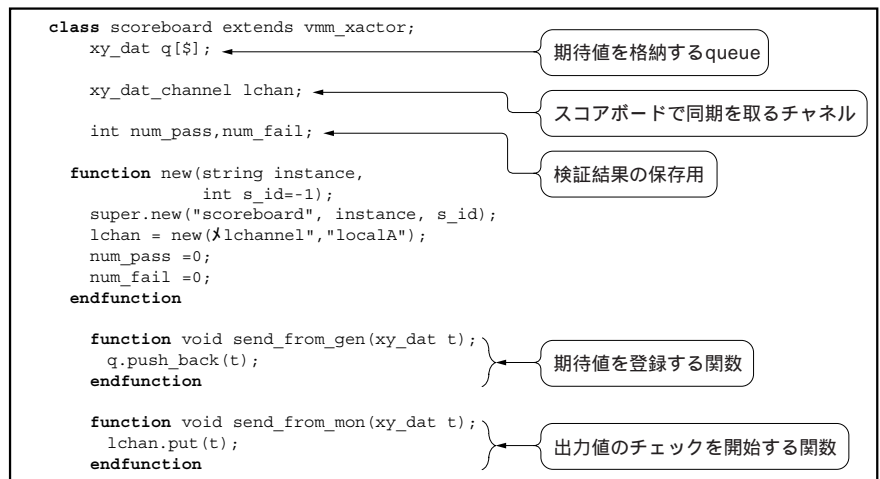
また，出力をチェックする関数send_from_monが呼ばれると，内部チャネルlchanにデータを送信（put）します．すると，**リスト4**のタスクmainのfor文の先頭でlchan.getを行い，関数send_from_monから送られたデータを受け取ります．チャネルからデータを受け取ると，キューから期待値を取り出し，チェックを行い，PASS/FAILの結果を更新します．

また，スコアボードには結果を出力する関数reportを作成しておきます．これにより，シミュレーション終了時にこの関数reportを呼び出すことで，シミュレーション結果を出力できます．こういったしくみを用意しておくことで，検証環境を改善していくことができます．

スコアボード
期待値を格納するキュー
期待値を登録する関数
出力値をチェックする関数
レポートを出力する関数
内部チャネル
main処理

図6　スコアボードの構成
期待値を格納し，期待値を登録し，出力値をチェックする機能だけでなく，レポートなどの機能を持たせておく．

リスト3
スコアボードの実現(1)
必要な変数の宣言，期待値を登録する関数，出力値をチェックする関数の実現部．

```
class scoreboard extends vmm_xactor;
    xy_dat q[$];

    xy_dat_channel lchan;

    int num_pass,num_fail;

  function new(string instance,
               int s_id=-1);
    super.new("scoreboard", instance, s_id);
    lchan = new(✗lchannel","localA");
    num_pass =0;
    num_fail =0;
  endfunction

    function void send_from_gen(xy_dat t);
      q.push_back(t);
    endfunction

    function void send_from_mon(xy_dat t);
      lchan.put(t);
    endfunction
```

リスト4
スコアボードの実現[2]
レポート関数や処理の中心となるmainタスクの実現．

```
virtual task main();
   fork
      super.main();
    join_none
    for (int i=0; i<20; i++) begin
      xy_dat   t, golden ;
      lchan.get(t);
      golden = q.pop_front();
      if ( t.kind == golden.kind ) num_pass ++;
      else                         num_fail++;
    end
  endtask

 virtual  function void report ( );
    int total;
    string str,str1,str2,str3;
    total= num_pass+num_fail;
    str1.itoa(num_pass);
    str2.itoa(num_fail);
    str3.itoa(total);
    str = { "report  pass=", str1,
          ", fail=", str2, ",  total count=", str3};
    `vmm_note(log, str);
  endfunction

endclass
```

チェックするデータがチャネルに送られたらここでgetし，期待値をキューから取り出す

判定を行い，PASS/FAILの結果を更新する

チェック結果をレポートするfunction

● コールバック用の仮想クラスの作成

次に，ドライバ/モニタの改造を行います．

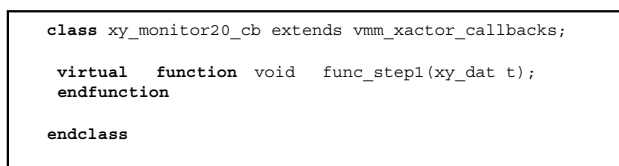
スコアボードにデータを登録するためには，ドライバ/モニタでコールバックを使えるようにしないといけません．そのためには，コールバック用の仮想クラスを作成します．ここではモニタ用のコールバックの仮想クラスを**リスト5**に示します．

トランザクタvmm_xactorで使用するコールバックをまとめるクラスは，vmm_xactor_callbacksを継承して作成します．この段階では，再度，どのようなコールバックを実装するか分からないので，virtualを付けた仮想関数としてfunc_step1を作成しました．複数のvirtual関数を作成しておくことも可能です．

● コールバック・ポイントを作成する

コールバックの仮想クラスができたら，ドライバとモニタにコールバックするためのしくみを組み込みます．

リスト5　モニタのコールバックの仮想関数
実体はないvirtualな関数func_step1を定義した．

```
class xy_monitor20_cb extends vmm_xactor_callbacks;

 virtual   function  void   func_step1(xy_dat t);
 endfunction

endclass
```

トランザクタを作成する段階では，どのようなものが作成されるか分からないので，トランザクタの記述中にコールバックするためのポイントを`vmm_callbackを用いて記述しておきます．モニタの場合には，**リスト6**のようにデータを受信した後に関数func_step1を呼び出すポイントを作成します．

この段階ではまだ関数func_step1は仮想であり実体はありませんが，このモニタをシミュレーションで使うことは可能です．この場合には，このコールバックはないものとして動作します．

● スコアボード用コールバック・クラスの作成

これから，機能を追加するコールバックのクラスを作成してみます．

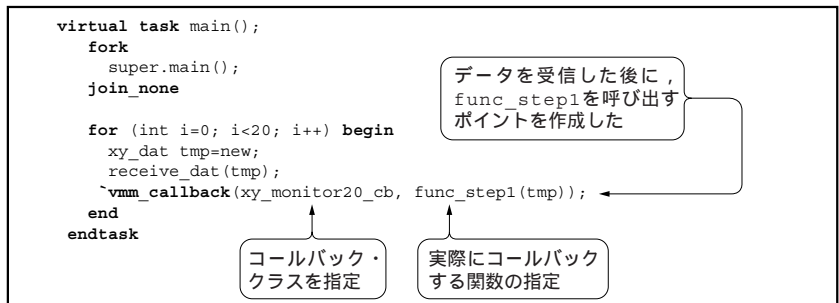
モニタのコールバックの仮想クラスはxy_monitor20_cbでしたが，スコアボードに接続するためのコールバック・クラスをmy1_monitir20_cbとして作成したのが**リスト7**になります．

仮想クラスで定義したvirtual関数と，関数名や引き数が同じであれば，自由に関数を作成することができます．モニタでは，スコアボードに出力値をチェックするための関数を呼び出しています．

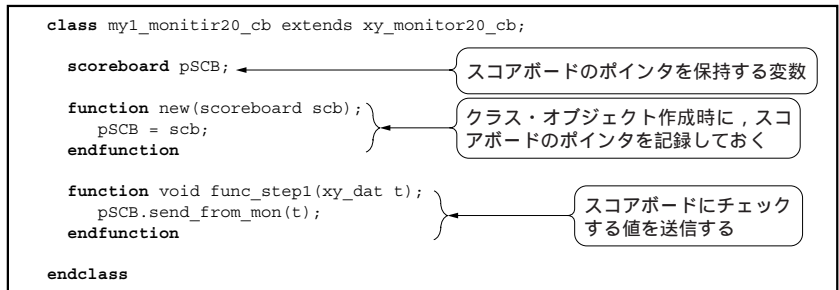
● vmm_envで変数定義

スコアボード，ドライバのコールバック，モニタのコー

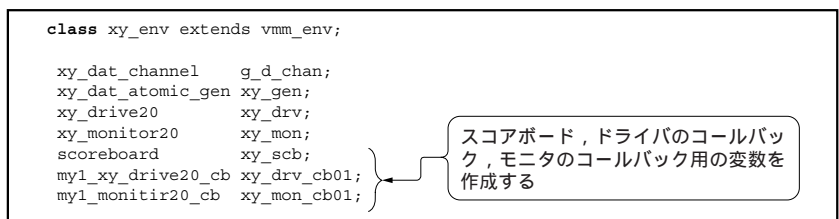
リスト6
モニタのタスクmainにコールバック・ポイント設定
`vmm_callbackを使って呼び出す関数を定義していく．

```
  virtual task main();
     fork
       super.main();
     join_none

     for (int i=0; i<20; i++) begin
       xy_dat tmp=new;
       receive_dat(tmp);
      `vmm_callback(xy_monitor20_cb, func_step1(tmp));
     end
   endtask
```


リスト7
モニタ用のコールバックの実体定義
スコアボードのためにスコアボードのポインタ（ハンドル）を保持し，関数func_step1が呼ばれたらスコアボードの関数send_from_monを呼び出す．

```
 class my1_monitir20_cb extends xy_monitor20_cb;

   scoreboard pSCB;

   function new(scoreboard scb);
      pSCB = scb;
   endfunction

   function void func_step1(xy_dat t);
      pSCB.send_from_mon(t);
   endfunction

 endclass
```


リスト8
xy_envで変数定義
スコアボード，コールバックに関する変数を定義する．

```
 class xy_env extends vmm_env;

  xy_dat_channel    g_d_chan;
  xy_dat_atomic_gen xy_gen;
  xy_drive20        xy_drv;
  xy_monitor20      xy_mon;
  scoreboard        xy_scb;
  my1_xy_drive20_cb xy_drv_cb01;
  my1_monitir20_cb  xy_mon_cb01;
```


ルバックはクラスなので，**リスト8**のように変数を定義しておきます．

● vmm_envでインスタンスを作成しコールバックを登録

VMMではインスタンスの作成は，vmm_envの関数build中で行うことになっています．そこで**リスト9**に示すように，スコアボード，ドライバのコールバック，モニタのコールバックのインスタンスを作成します．その後，コールバックをドライバ/モニタに登録します．ドライバやモニタというトランザクタにコールバックを登録するためにはappend_callbackを使います．引き数としては，第1引き数にコールバックのクラス名，第2引き数に呼び出す関数を与えることで，コールバックとして登録することができます．

また，append_callbackを繰り返し呼び出すことで，複数のコールバックを登録することも可能です．

● vmm_envでスコアボード・クラスを起動

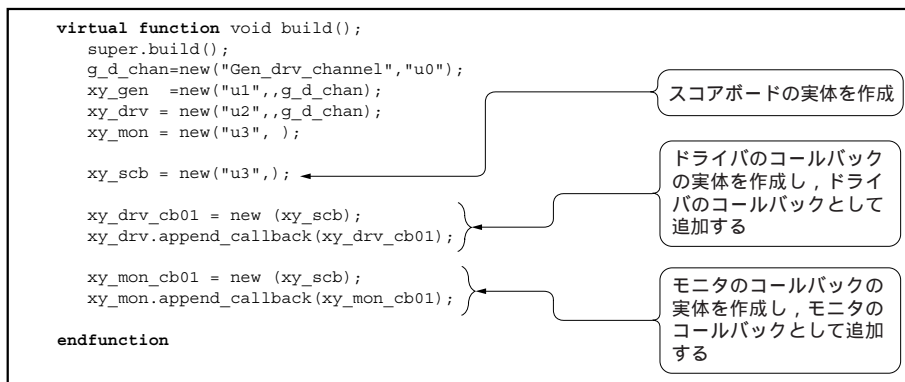
今回作成したスコアボードはvmm_xactorをベースに作成しているので，vmm_envのタスクstartで，スコアボードxy_scbのstart_xactorを呼び出すことで，スコアボードを動作させます（**リスト10**）．同様にvmm_envのタスクstopでは，stop_xactorを呼び出し，スコアボードを停止させます．

● スコアボードのreportの呼び出し

ここまで，VMMでランダム生成と自動チェックを使ってきましたが，シミュレーションの終わりで一番大事なことを行います．それは，レポートです．ランダム生成を使い，スコアボードで検証したときに，実際にどのような検証が行われたのかを設計者・検証者にフィードバックする必要があります．

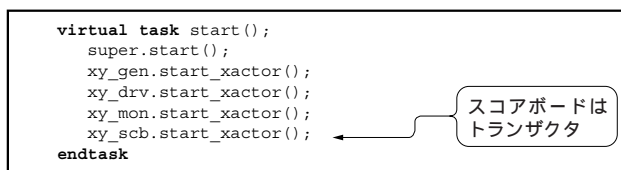
今回は，PASSしたパターン数とFAILしたパターン数

リスト9
xy_envの関数buildの設定
スコアボード，コールバックのインスタンス，および，コールバックを登録する．

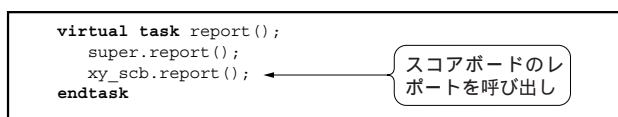
```
virtual function void build();
   super.build();
   g_d_chan=new("Gen_drv_channel","u0");
   xy_gen  =new("u1",,g_d_chan);
   xy_drv = new("u2",,g_d_chan);
   xy_mon = new("u3", );

   xy_scb = new("u3",);

   xy_drv_cb01 = new (xy_scb);
   xy_drv.append_callback(xy_drv_cb01);

   xy_mon_cb01 = new (xy_scb);
   xy_mon.append_callback(xy_mon_cb01);

endfunction
```


リスト10　xy_envのタスクstartの設定
スコアボードはvmm_xactorであるので，起動させる必要がある．

```
virtual task start();
   super.start();
   xy_gen.start_xactor();
   xy_drv.start_xactor();
   xy_mon.start_xactor();
   xy_scb.start_xactor();
endtask
```

スコアボードはトランザクタ

リスト11　xy_envのタスクreportの設定
シミュレーション終了時に，スコアボードのレポートを呼び出し，検証結果を出力する．

```
virtual task report();
   super.report();
   xy_scb.report();
endtask
```

スコアボードのレポートを呼び出し

を示すようにしています．これを，vmm_envのタスクreportで行うようにします．スコアボードにはレポート用の関数reportを用意しているので，**リスト11**のようにvmm_envのタスクreportを作成します．

これにより，ランダム生成を使いテスト・パターンを大量に生成し，その大量のテスト・パターンに対する出力値をスコアボードで確認し，その結果を設計者・検証者に見せることができます．

このような検証環境を作成するのがめんどうに思われるかもしれませんが，一度作成してしまえば自動で検証が進んでいくため，多くのテスト・パターンを生成するためには効率的です．

あかぼし・ひろき
（株）ソリューション・デザイン・ラボラトリ

<筆者プロフィール>
赤星博輝．ハードウェアの検証とソフトウェアのテストの融合が現在のテーマです．ハードウェアではVerification Methodology ManualとSystemVerilogを推進し，ソフトウェアではRTOSを中心に活動中です．